REMARKS

Claims 1-22 are pending. All claims were rejected under 35 U.S.C. § 102(b) and/or 35 U.S.C. § 103(a). These rejections are traversed.

Rejections under 35 U.S.C. § 102(b)

Claims 1, 3-6, 9-10, 12-15 and 18-22 were rejected under 35 U.S.C. § 102(b) as being anticipated by McLennan, Michael J., "Object-oriented Programming with [incr Tcl] Building Mega-Widget with [incr Tk]". To establish a prima facie case for anticipation under 35 U.S.C. § 102(b), the cited reference must teach every aspect of the claimed invention either explicitly or impliedly. For the reasons discussed below, it is respectfully submitted that the Examiner has not established a prima facie case under 35 U.S.C. § 102(b) for any of the present claims, and that therefore, the present claims are allowable.

The invention generally relates to a technique for optimizing the handling of changes to option values. The present invention provides an option data structure and a mapping data structure that enables option-change handling code to be identified efficiently when an option-change notification occurs. An object is defined with an option data structure, which supports references to options without having to preallocate memory for the full option values. The option data structure can be, for example, a linked list. The invention uses a mapping data structure, which preferably maps an option name and class to an option binding. The mapping data structure can be, for example, a hash table. The option binding identifies a change handler that notifies objects that are affected by a change in an option value.

With the option data structure of the present invention, the potential option values for an object are not preallocated storage space. This can result in substantial savings of memory space, when there is a very large set of possible full option values associated with the object or class that are seldom set on the object. Storage space is only allocated when the default value for an option value is modified for the object in contrast to prior art methods that allocate memory for an attribute even when the object inherits a value for the attribute from an object up the hierarchy. Further, by pre-computing mappings of option names and class to an option binding, a change in an object's option value can be effected seamlessly, without having to iterate through the object's or class's property names to determine which property is affected by the change. Thus, the use of the claimed option and mapping data structures makes it possible to define a

very large set of possible options for a class without the time and storage space limitations associated with the prior art.

The background section of the Applicants' specification discusses prior art techniques for storing property values that relate to the [incr Tk] language. In [incr Tk], there is preallocation of memory space for option values and the specification has now been amended to correct any reference to the contrary. Because [incr Tk] does preallocate memory space for option values, and the claimed invention does not, the rejections based on [incr Tk] should be withdrawn.

[incr Tcl] is a Tcl extension that adds object-oriented programming constructs to Tcl. [incr Tk] builds on top of a graphics toolkit called Tk, so when an application is built using [incr Tk], [incr Tk] "mega-widgets" may be used which may in turn contain other [incr Tk] mega-widgets, and may also contain basic Tk widgets. Tk defines the basic widgets such as text-entry fields and scrollbars, while the [incr Tk] mega-widgets are larger-scale entities such as file choosers or color choosers.

The construction of a mega-widget is discussed in Mc Lennan. (*See* Pages 76-85.) As shown in Figs. 2-8, each mega-widget includes a set of configuration options. A configuration option can have the same value as the option on a master list or can be set just for the mega-widget. Each widget class has a default set of options that have the same value as the option on a master list. The default set of options is called the usual option-handling code. For example, the foreground, background, font and cursor options in a label component have the same values as the options on the master list. (*See* Page 83, lines 5-35.)

Therefore, it is pertinent to look at the storage requirements both of Tk widgets and [incr Tk] mega-widgets. These are discussed in the manual pages for each release of Tk. The manual page for the basic TK widget in release 8.0.x (released between August 1997 and March 1999) is available at: http://www.tcl.tk/man/tcl8.0/TkLib/ConfigWidg.htm. Referring to the manual page, Tk_ConfigureWidget processes a table specifying configuration options that are supported. Each entry in the table is a Tk_ConfigSpec structure which includes a type field (type of configuration option, for example, color) and an offset field which indicates where in the record to store information about this option.

Toward the bottom of this page, under the heading "TK_OFFSET", there is the following paragraph:

> "The Tk_Offset macro is provided as a safe way of generating the offset values for entries in Tk_ConfigSpec structures. It takes two arguments: the name of a type of record, and the name of a field in that record. It returns the byte offset of the named field in records of the given type."

From this paragraph (and the surrounding context), it can be inferred that every option value that can be set on a Tk widget has a unique, pre-allocated location in the widget's "record" where that option value will be stored. In addition, to the above-paragraph describing the Tk_Offset macro, the man page also describes a Tk_ConfigureWidget macro that includes "a collection of command-line arguments (argc and argv) to fill in fields of a record (widgRec)." It can be inferred that in order to fill in the fields, the fields must be pre-allocated. Furthermore, the Tk_ConfigSpec structure that specifies a configuration option includes an offset field which "indicates where in widgRec to store information about this option". If space were allocated only for options that are actually set, it would not be possible to specify a fixed offset in the record for storing a value simply based on knowing the type of record and the name of a field. It would also be necessary to take into account the other options that had already been set on the specific record in question.

The Archetype is the base class for all [incr Tk] mega-widgets. The man page for Archetype in version 3.1 of [incr Tk] (released May 29, 1999) is available on the Internet at http:// www.tcl.tk/man/itcl3.1/Archetype.n.html. The "itk_component add" method is used to add a component widget to a mega-widget. In calling this method, the "keep" command (documented under "itk_component add") is used to specify option names of the component widget that should be connected to option names of the mega-widget. Each option name to be "kept" is specified individually, and later on if one of these options is set on the mega-widget, the [incr Tk] mechanisms will notice that and will also set the same option value on the component widget. Since each mega-widget instance will have its own collection of component widget instances, it must be the case that each mega-widget instance builds up its own table of "kept" options, whose size is (at a minimum) proportional to the number of options that could be set on the mega-widget, regardless of how many of those options are actually set on the mega-widget.

The "itk_option define" command provides a mechanism for defining option names on

mega-widgets that might not correspond to option names on any of the component widgets. The arguments to this command include an "init value" for which the following explanation is provided: "The init value string is used as a last resort to initialize the option if no other value can be used from an existing option, or queried from the X11 resource database." The discussion of "initializing" the option value from the "init" argument provides strong evidence that there is already a preallocated memory location to hold the option value: given the existence of this location, it must be filled with some value, so the init value is used as a last resort for this purpose. If a preallocated location did not already exist, there would be no discussion of the need for "initializing". In contrast, there may be a discussion regarding use of default value if the location was not allocated.

Thus, McLennan does not discuss the claimed technique for defining an object with an option data structure that supports references to option values without preallocation of memory for full option values. It appears that McLennan is consistent with traditional object-oriented techniques, as McLennan does not suggest modifying them in any way. As a result, a preallocated field is provided for each of the configuration options when the object is created, even if the option is never set on the object. Even when this option is not set, it still occupies its own dedicated memory space in the object.

McLennan does not discuss or disclose any technique for avoiding the memory preallocation for configuration options. As such, McLennan does not relate to the claimed technique for defining an object with an option data structure that supports references to option values without preallocation of memory space for the full option values.

Rejections under 35 U.S.C. § 103(a)

Claims 7-8 and 16-17 were rejected under 35 U.S.C. § 103(a) as being unpatentable over McLennan in view of Hostetter et al., "Curl: A Gentle Slope Language for the web," World Wide Web Journal, Spring, 1997. These rejections are traversed.

Claims 2 and 11 are rejected under 35 U.S.C. § 103(a) as being unpatentable over McLennan in view of Li et al. (U.S. Patent No. 5,943,496). These rejections are traversed.

Cited prior art Li discusses identifying an object instance associated with a particular instance name.. An object name table stores a name with a component object instance. Li's discussion of an object name table which associates names to objects does not teach or suggest the Applicants' claimed "mapping data structure" which stores mapping to an option binding. Li does not teach or suggest an object with an option data structure which supports references to option values without preallocation of memory space and thus does not teach or suggest option bindings. Thus, there is no teaching or suggestion of the Applicants' claimed "mapping data structure". The Applicants' claimed mapping data structure maps option values without preallocation of memory space that are included in the object. Li merely associates a name with an object.

As claim 2 depends directly from claim 1, it also contains all of the limitations of base claim 1. Similarly, as claim 11 depends directly from base claim 10, it contains all of the limitations of that base claim.

As claims 7-8 depend either directly or indirectly from claim 1, they also contain all of the limitations of base claim 1. Similarly, as claims 16-17 depend either directly or indirectly from base claim 10, they contain all of the limitations of that base claim.

For the reasons set forth above, Claims 1 and 10 and their respective dependent claims are in condition for allowance. Reconsideration of the rejections of claims 2, 7-8, 11 and 16-17 under 35 U.S.C. § 103(a) is respectfully requested.

Information Disclosure Statement

A Supplemental Information Disclosure Statement (SIDS) is being filed concurrently herewith. Entry of the SIDS is respectfully requested.

CONCLUSION

In view of the above remarks, it is believed that all claims are in condition for allowance, and it is respectfully requested that the application be passed to issue. If the Examiner feels that a telephone conference would expedite prosecution of this case, the Examiner is invited to call the undersigned.

Respectfully submitted,

HAMILTON, BROOK, SMITH & REYNOLDS, P.C.

By Caroline M Fleming

Caroline M. Fleming
Registration No. 45,566
Telephone: (978) 341-0036
Facsimile: (978) 341-0136

Concord, MA 01742-9133
Dated: 9/8/04